Ex4 Tutorial - Forward and Back-propagation

Subscribe for email updates. PINNED UNRESOLVED

No tags yet. + Add Tag

Sort replies by: Oldest first Newest first Most popular

Tom Mosher COMMUNITY TA · 2 days ago PINNED

(Work in progress... if you use this tutorial, please provide feedback for errors and suggestions.)

This tutorial outlines the process of accomplishing the goals for Programming Exercise 4. The purpose is to create a collection of all the useful yet scattered and obscure knowledge that otherwise would require hours of frustrating searches.

This tutorial is targeted solely at vectorized implementations. If you're a looper, you're doing it the hard way, and you're on your own.

I'll use the less-than-helpful greek letters and math notation from the video lectures in this tutorial, though I'll start off with a glossary so we can agree on what they are. I will also suggest some common variable names, so students can more easily get help on the Forum.

It is left to the reader to convert these lines into program statements. You will need to determine the correct order and transpositions for each matrix multiplication.

Most of this material appears in either the video lectures, slides, course wiki, or the ex4.pdf file, though nowhere else does it all appear in one place.

Glossary:

Each of these variables will have a subscript, noting which NN layer it is associated with.

Θ: A matrix of weights to compute the inner values of the neural network. When we used single-vector theta values, it was noted with the lower-case character θ.

z : is the result of multiplying a data vector with a Θ matrix. A typical variable name would be "z2".

a : The "activation" output from a neural layer. This is always generated using a sigmoid function $g()$ on a z value. A typical variable name would be "a2".

δ : lower-case delta is used for the "error" term in each layer. A typical variable name would be "d2".

Δ : upper-case delta is used to hold the sum of the product of a δ value with the previous layer's a value. In the vectorized solution, these sums are calculated automatically though the magic of matrix algebra. A typical variable name would be "Delta2".

Θ gradient : This is the thing we're looking for, the partial derivative of theta. There is one of these variables associated with each Δ. These values are returned by nnCostFunction(), so the variable names must be "Theta1_grad" and "Theta2_grad".

$g()$ is the sigmoid function.
$g'()$ is the sigmoid gradient function.

Tip: One handy method for ignoring a column of bias units is to use the notation SomeMatrix(:,2:end). This selects all of the rows of a matrix, and omits the entire first column.

Nearly all of the editing in this exercise happens in nnCostFunction.m, unlike the previous exercises. Let's get started.

A note regarding bias units, regularization, and back-propagation:
There are two methods for handing the bias units in the back-propagation and gradient calculations. I've described only one of them here, it's the one that I understood the best. Both methods work, choose the one that makes sense to you and avoids dimension errors. It matters not a whit whether the bias unit is dropped before or after it is calculated - both methods give the same results, though the order of operations and transpositions required may be different. Those with contrary opinions are welcome to write their own tutorial.

Forward Propagation:
We'll start by outlining the forward propagation process. Though this was already accomplished once during Exercise 3, you'll need to duplicate some of that work because computing the gradients requires some of the intermediate results from forward propagation.

1 - Expand the 'y' output values into a matrix of single values (see ex4.pdf Page 5). This is most easily done using an eye() matrix of size num_labels, with vectorized indexing by 'y', as in "eye(num_labels)(y,:)". Discussions of this and other methods are available in the Course Wiki - Programming Exercises section. Typical variable name would be "y_matrix". (**Update**: Deleted incorrect reference to Ex3, added eye() and Wiki references).

2 - perform the forward propagation:
a_1 equals the X input matrix with a column of 1's added (bias units)
z_2 equals the product of a_1 and Θ_1
a_2 is the result of passing z_2 through $g()$
a_2 then has a column of 1st added (bias units)
z_3 equals the product of a_2 and Θ_2
a_3 is the result of passing z_3 through $g()$

Cost Function, non-regularized
3 - Compute the unregularized cost according to ex4.pdf (top of Page 5), using a_3, your y_matrix, and m (the number of training examples). Cost should be a scalar value. If you get a vector of cost values, you can sum that vector to get the cost.
Update: Remember to use element-wise multiplication with the log() function.

Now you can run ex4.m to check the unregularized cost is correct, then you can submit Part 1 to the grader.

Cost Regularization

4 - Compute the regularized component of the cost according to ex4.pdf Page 6, using Θ_1 and Θ_2 (ignoring the columns of bias units), along with λ, and m. The easiest method to do this is to compute the regularization terms separately, then add them to the unregularized cost from Step 3.

You can run ex4.m to check the regularized cost, then you can submit Part 2 to the grader.

Sigmoid Gradient and Random Initialization
5 - You'll need to prepare the sigmoid gradient function $g'()$, as shown in ex4.pdf Page 7

You can submit Part 3 to the grader.

6 - Implement the random initialization function as instructed on ex4.pdf, top of Page 8. You do not submit this function to the grader.

Backpropagation
7 - Now we work from the output layer back to the hidden layer, calculating how bad the errors are. See ex4.pdf Page 9 for reference.

δ_3 equals the difference between a_3 and the y_matrix.
δ_2 equals the product of δ_3 and Θ_2 (ignoring the Θ_2 bias units), then multiplied element-wise by the $g'()$ of z_2 (computed back in Step 2).

Note that at this point, the instructions in ex4.pdf are specific to looping implementations, so the notation there is different.
Δ_2 equals the product of d_3 and a_2. This step calculates the product and sum of the errors.
Δ_1 equals the product of d_2 and a_1. This step calculates the product and sum of the errors.

Gradient, non-regularized
8 - Now we calculate the gradients, using the sums of the errors we just computed. (see ex4.pdf bottom of Page 11)
Θ_1 gradient equals Δ_1 scaled by $1/m$
Θ_2 gradient equals Δ_2 scaled by $1/m$

The ex4.m script will also perform gradient checking for you, using a smaller test case than the full character classification example. So if you're debugging your nnCostFunction() using the "keyboard" command during this, you'll suddenly be seeing some much smaller sizes of X and the Θ values. Do not be alarmed.

If the feedback provided to you by ex4.m for gradient checking seems OK, you can now submit Part 4 to the grader.

Gradient Regularization
9 - For reference see ex4.pdf, top of Page 12, for the right-most terms of the equation for $j >= 1$.

in Step 8, you have already calculated the non-regularized gradient. Now you will calculate the regularization terms for each theta gradient:
$(\lambda/m) * \Theta_1$ (omit the column of bias units)
...and
$(\lambda/m) * \Theta_2$ (omit the column of bias units)
and add these regularization terms to the appropriate Θ_1 gradient and Θ_2 gradient terms found in

Step 8.
Note: there is an errata in the lecture video and slides regarding some missing parenthesis for this calculation.
The ex4.pdf file is correct.

The ex4.m script will provide you feedback regarding the acceptable relative difference.
If all seems well, you can submit Part 5 to the grader.

Now pat yourself on the back.

17 · flag

Tom Mosher COMMUNITY TA · 21 hours ago PINNED

In debugging your nnCostFunction(), I strongly recommend using Apurva's test cases at this thread:
https://class.coursera.org/ml-005/forum/thread?thread_id=1783#post-7870

1 · flag

Hatice Mujde Sari · 38 minutes ago

Tom,

Thanks so much. Your tutorial helped a lot. Very clear and useful

1 · flag

Tom Mosher COMMUNITY TA · 7 minutes ago

Thanks for the feedback.

0 · flag

+ Comment

Vimal Kumar · 2 days ago

for calculating delta2, we need to delts3 and theta2' (ignoring the bias units).

i was confused at this point itself, because if we dont ignore ths bias units, the matrix dimensions dont match. but the exercise handout and also the class notes dont mention this point at all.....
just shows delta2=(theta2'*delta3).*(g'(z2))

and similarly for delta3 etc.

isn't it misleading in the text material of the course?

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

Thanks Vimal,
I will update the notes to clarify this.

Update:
Omitting the bias units is mentioned in ex4.pdf on page 9, step 4, though it is hidden by referencing δ_0. This is the second method of handling the bias units, as mentioned in the tutorial.

0 · flag

Vimal Kumar · 2 days ago

but even before the location ex4.pdf page 9, step 4, in order to calculate delta2 in step 3 itself, we need to ignore the bias term in the equation
delta2=(theta2'*delta3).*(g'(z2)) otherwise dimensions dont match
theta2' = 26 x 10
delta3 = 10 x 5000 (vectorized method)
theta2'*delta3 = 26 x 5000
g'(z2) = 25 x 5000

so a mismatch when we perform the .* operation

it can be resolved if we ignore the bias term in the step 3 itself, then delta2 becomes 25 x 5000

so the omitting of bias term should be mentioned in step 3 and also need to mention in lecture notes
https://class.coursera.org/ml-005/lecture/51

thanks,

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

Hi Vimal,
Thanks for your comments on this tutorial, it's very helpful.

In my implementation, I drop the bias units from Theta2, and use a different operator order and transposition, when it is used to calculate d2, thereby avoiding the dimensions problem. The result is the same whether the bias unit is dropped before or after the product it is calculated, the methods are equivalent. This is the 'second method' that I mention in the notes. Perhaps I should highlight that in bolder text, so it's clear that I'm only presenting one of two equivalent methods.

Thanks!

0 · flag

Euphrates Zeray Asfaha · 13 hours ago

Thanks Tom,

I spent a lot of time trying to identify why my answers were wrong although everything seemed to be ok. Now, it worked perfect but I still do not understand where we get the bias

for the Theta's. We have added bias to the Units

0 · flag

Tom Mosher COMMUNITY TA · 6 hours ago

Hi Euphrates,
Like every theta value, the one for the bias unit is initialized to a small random value, then it is adjusted by the fmincg() function (which runs a gradient descent algorithm) such that the total cost function is minimized. All values of theta emerge from the ground spontaneously in this way - not just the one for the bias unit.

The bias units themselves (the 1's that are added) simply provide a coefficient to multiply the theta(1) value by. Taken together, the bias unit and theta(1) act similarly to the "y-intercept" in the equation for line.
Take this form of a line equation:

$$y = mx + b$$

Now re-write it as:

$$y = b + mx$$

In matrix form, that's equivalent to:

$$y = \begin{bmatrix} b & m \end{bmatrix} * \begin{bmatrix} 1 & x \end{bmatrix}^T$$

If we define $\theta = [b; m]$ and $x = [1 \quad x]$, it becomes $y = \theta^T x$. That should be very familiar by now.

For the matrix multiplication to work, the theta(1) value must have an element in X(1,:) to multiply by. A '1' does the trick.

0 · flag

+ Comment

Vimal Kumar · 2 days ago

i just noticed the post is tagged with 'julialang'.

i didn't understand how is julia lang related to this post? and there are many posts tagged as 'julialang'

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

That's odd, I don't know how that got there. I've deleted that tag.
(Perhaps she is tagging posts for her own reference) - **or I'm a hopelessly out of touch (see next post...)**

0 · flag

Vimal Kumar · 2 days ago

perhaps it related to this
http://julialang.org/

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

That seems likely. Someone is promoting their favorite tool on the forums.

0 · flag

+ Comment

Scott Francis · 2 days ago

I seem to be having some trouble regularizing the gradients. I know I'm computing the backprop and unregularized properly due to the example script and have "passed" part 4. It seems like to add the regularlization to the gradients (columns 2:end in the Theta?_grad matrices) is a fairly simple operation.

But the script is showing big scale differences. What am I missing? The regularization is just a scaling of each term right?

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

Hi Scott,
First, please delete the code listing from your post. That's not allowed under the Honor Code.

Second, verify that you're calculating the theta gradient based on theta (not theta gradient).

0 · flag

Scott Francis · 2 days ago

thanks Tom. Both for the honor reminder and pointing me in the right direction. That was one of those things I was just staring at and couldn't find.

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

No problem, I've been there plenty of times.

0 · flag

+ Comment

Kara Fulcher · 2 days ago

Tom, many thanks!
I had already submitted the first exercises and they passed, so I am guessing that these are not my problem. I have executed Backprop just as Tom describes here and am able to get my dimensions to match. (Though, Tom, does Theta#_grad reinsert a bias column? It looks to me like this is the case.

Is this my error?) However, when ex4 checks backpropagation, values 1-5 and 21-23 fail, with the values on the right all showing up as 0.
Values in the left column:
1) -0.00928
2) 0.00890
3) -0.00836
4) 0.00763
5) -0.00675
21) 0.31454
22) 0.11106
23) 0.09740
No matter what I tinker with, I end up with these 8 values and an unacceptably high variance of .407869. I notice in training the neural network that the cost goes quite low and then pops up high and never returns to its lowest point. Ultimately, I consistently get a training accuracy ~95%.
Has anybody else had a similar problem? I'm really stumped.

1 · flag

Tom Mosher COMMUNITY TA · 2 days ago

No, Theta Gradient does not insert a bias column.
If cost starts to minimize and then reverses direction and blows up, you've got some sort of structural problem in the code.

One problem with specifying what sizes the variables should be is that ex4.m invokes several different test cases. If you just put a "keyboard" statement in nnCostFunction(), you have to keep typing "return" to continue the program until the one you're looking for comes up.

I'll give it a try here this evening, specifying the variable sizes the first time the nnCostFunction is called, by putting a "keyboard" command just before nnCostFunction() returns. Stand by for updates (gotta go cook dinner for the family).

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

Sizes of variables, running ex4.m, stopping on the first pass through nnCostFunction() - by adding a "keyboard" command just before the end of the function:
(Note: these sizes apply to the method outlined in this tutorial. If you're using a different method, you could have different sizes).
a1: 5000x401
z2: 5000x25
a2: 5000x26
a3: 5000x10
d3: 5000x10
d2: 5000x25
Delta1 and Theta1_grad: 25x401
Delta2 and Theta2_grad: 10x26

2 · flag

Adam Sass · a day ago

Hi Tom!
I've read some of your posts, (thanks for them!) but i'm confused a little bit.
If you say it is only one method to ignore the bias term at computing d2, and not later. How could you make the product if you want to ignore it later, because in g'(a1*Theta1') theta is 25X401? And if you ignore it already at d2 (step 3) how come you write Theta1_grad nad Delta1 is 25x401, and Delta2 and Theta2_grad is 10x26?

Thanks,
Adam

0 · flag

Tom Mosher COMMUNITY TA · a day ago

Sorry. The method I described works for me, and I don't have any answers about how to make the other method work.

0 · flag

Adam Sass · a day ago

I found the problem, thanks, i could also make this method work!

1 · flag

Kyle DeRosa · a day ago

Hi Kara,

I had basically the same exact problem as you (successful matrix matching, but a relative difference of approx .4078).

What worked for me was looking at my calculations for Δ_1 and Δ_2:

$\Delta_1 = \delta_2 * a_1$

I was incorrectly stripping the bias terms from a_1 :

```
Delta_1 = delta_2 * a_1(:,2:end);
```

I changed my implementation to include a_1's bias term:

```
Delta_1 = delta_2 * a_1;
```

and my relative difference (without regularization) became much lower (approx. 2.2882e-11).

So if I've got right, it's important to remove the bias term δ_0 from either $\Theta^{(l)}$ or $\delta^{(l)}$, but it's

important to retain the bias term in $a^{(l)}$.

The intuition eludes me. My best guess is that while there's no way to update any bias node value (we always set $a_0^{(l)}$ to 1), we still want to be able to update its corresponding $\Theta_0^{(l)}$.

Hope it helps!

Kyle

0 · flag

Tom Mosher COMMUNITY TA · a day ago

Hi Kyle,
Your best-guess is correct. It does no good to have a bias term unless we can just its theta coefficient, to control how much bias gets applied.

0 · flag

Kara Fulcher · 21 hours ago

Kyle, thanks for the insight. I had come to this conclusion, too. You get so blind to something you've written after a while. I started from scratch and immediately recognized my error. As Valentin said in a different thread: don't mess with the bias units people!

I completely appreciate your efforts and patience, Tom.

I do wonder why Ng warns so strongly against the vectorized approach. I've got six lines of code for backprop and another two for the regularization -- and I only looked at Octave for the first time when this class began. (I don't know any other coding, except a little R, for that matter.)

Thanks for the support! It makes a huge difference!

1 · flag

Attila Szász · 4 hours ago

Tip:

If your code runs correctly but your values are far off you might need to transpose your Delta1 and Delta2 before calculating Theta1_grad and Theta2_grad as they get unrolled later and the order of elements will matter.

I spent a good hour on this, hopefully nobody else will have to. :-)

0 · flag

+ Comment

Bhanu Krishna · 2 days ago

tom,
without any for loops, only vectorization.
a3=h=5000*10
y=5000*1
after this not getting what to do.
according to your post above i should be converting y into a y_matrix of 5000*10? this part confused.

0 · flag

Tom Mosher COMMUNITY TA · 2 days ago

The feed-forward process is the same as you used in Exercise 3 - in your predict.m function. The Week 4 videos may also be helpful.

0 · flag

Bhanu Krishna · 2 days ago

The second step of forward propagation no problem. As you said it is same as predict.m in exercise 3.Got h as 5000*10 matrix.
I am struggling with first step, that of expanding y=5000*1 to y_matrix=5000*10.

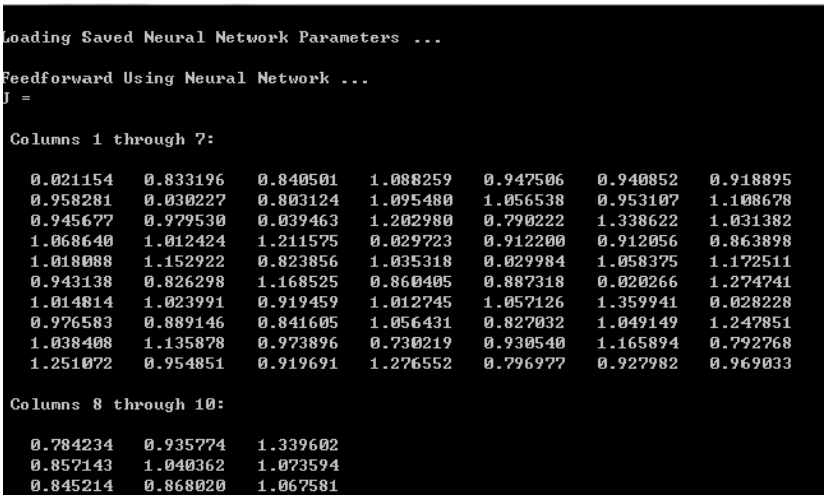
0 · flag

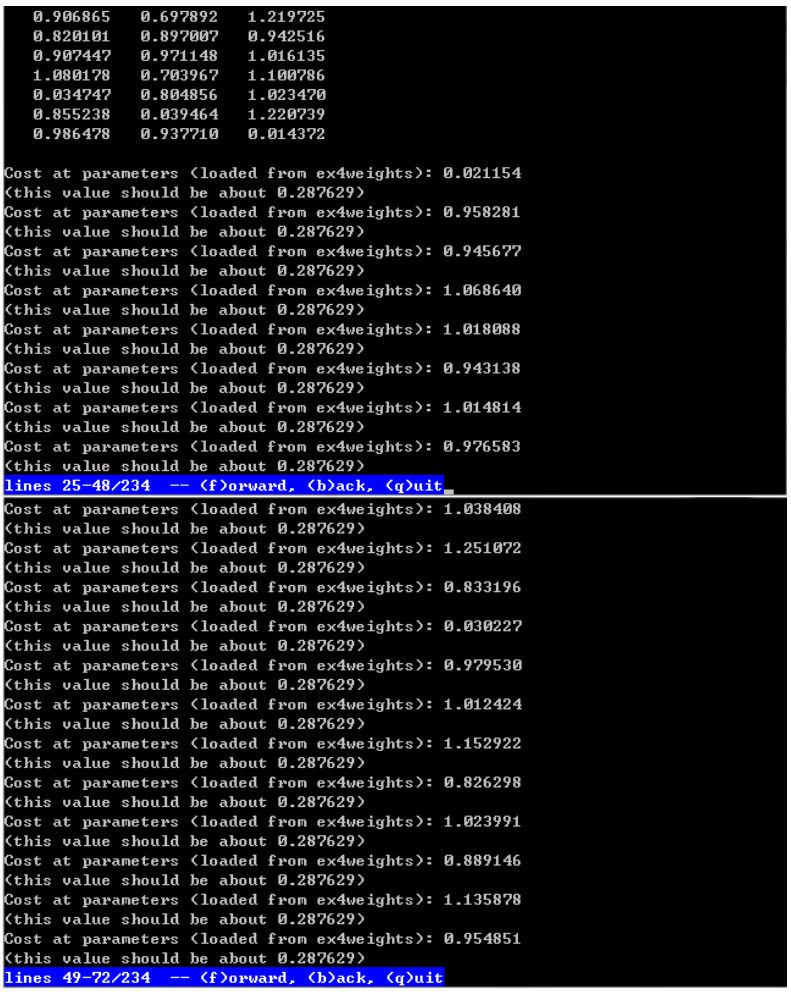
Tom Mosher COMMUNITY TA · 2 days ago

Search the forum, you'll find solutions like this:
https://class.coursera.org/ml-005/forum/thread?thread_id=1799#comment-4251
and this:
https://class.coursera.org/ml-005/forum/thread?thread_id=2091

0 · flag

sanghyun yuk · a day ago





Dear Tom,

Although none of the cost is exactly 0,287629, the cost seems to be in the ballpark. Is this a correct implementation? Or am I doing something wrong? Can you please enlighten me? Thank you!

0 · flag

Tom Mosher COMMUNITY TA · a day ago

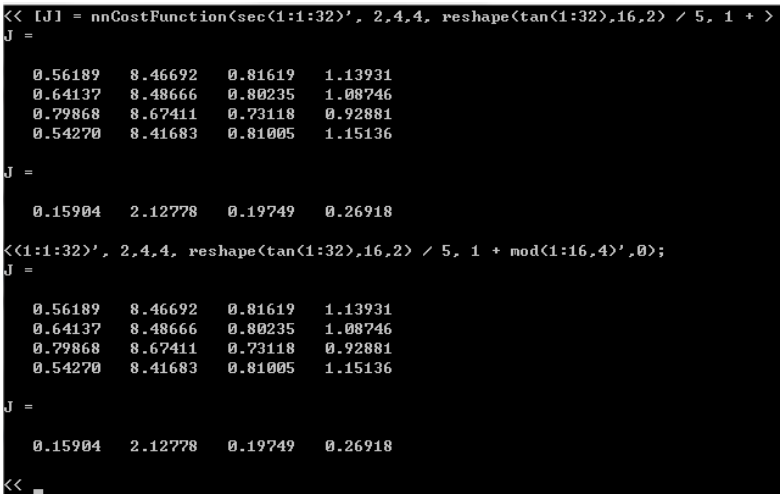
Looks to me like there's a problem. The display of the 10x10 matrix isn't expected, and I'm not sure why it is repeatedly displaying the "cost at parameters..." lines. That should only happen once, then the screen should pause until you "press enter to continue".

So, your ex4.m file may have been accidentally modified.
You're not getting the correct cost value from nnCostFunction(), either.

I recommend you try the a unit test for nnCostFunction() at this thread.
https://class.coursera.org/ml-005/forum/thread?thread_id=1783#post-7870
Post your results here, and see if the problem becomes clearer.

0 · flag

sanghyun yuk · 21 hours ago



looks like there is some serious problem? why am i not getting a single value for J cost...

0 · flag

sanghyun yuk · 19 hours ago

Dear Tom,
is there any way i can show my cod to you without breaking the honor code. i don't understand why im getting the matrix like u said, much less the incorrect J value..

0 · flag

Tom Mosher COMMUNITY TA · 18 hours ago

You have interesting results. That 4x4 matrix could have come from using the log() function with matrix multiplication. It should be done with element-wise multiplication, so the size of the result doesn't change. Please check that, and send me a reply.

0 · flag

A post was deleted

Tom Mosher COMMUNITY TA · 18 hours ago

The issue is with the type of multiplication operator - matrix multiply, vs. element-wise multiply. Please delete your line of code - it is part of the exercise solution.

0 · flag

sanghyun yuk · 16 hours ago

Tom,

ok. now i understand the element wise multiplication part, but i'm getting a parsing error for this code: J_theta = sum(J) / m . I calculated J first and then tried to sum it all and divide by m again. what can be a problem this time?

0 · flag

Tom Mosher COMMUNITY TA · 15 hours ago

Can you provide the error code?

0 · flag

sanghyun yuk · 15 hours ago

do i have to sum twice and then divide by m? now im getting the error saying i have two incompatible operands 10 X 5000 and 5000 X 10. is this because of element wise mutliplicaiton?

0 · flag

sanghyun yuk · 14 hours ago

Actually. I figured it out. Thank you so much for your guidance. Tom. Basically I didn't really grasp the difference between matrix multiplicaiton and element wise multiplication and summing afterwards. now I'm tackling the next part of Q1.

1 · flag

Tom Mosher COMMUNITY TA · 14 hours ago

Good work!

0 · flag

+ Comment

Rob van Putten · a day ago

wow.. this is what I needed.. I finally managed to solve the exercise, many thanks for your help!

(BTW, for those still struggling these are approx. values you should expect, I needed 50 iterations for Neural Net Gradient Function (Backpropagation))
iteration 1: cost 3.32
...
iteration 50: cost 5.75E-1, which gives an accuracy of about 95.28

And for Regularized Gradient

iteration 1: cost 3.34
...
iteration 50: cost 4.49E-1, which gives an accuracy of about 96.14

0 · flag

Tom Mosher COMMUNITY TA · a day ago

Hi Rob,
I'm glad you found the material helpful.
Note that 50 iterations is the default number that is set in ex4.m. The Optional part of ex4.pdf (Page 13) recommends that you adjust this number and lambda, to see the impact on cost and training accuracy from using different values.

0 · flag

+ Comment

Jakub Prüher · a day ago

I'm getting frustrated with this exercise!

I understand the forward propagation process, I understand why the dimensions are what they are, but I can't for the life of me calculate the correct cost.

The cost value I'm getting is 5.875588, instead of the 0.287629. At first I used one-liner to calculate the cost (utilizing **sum of sums** and some element-wise matrix multiplications). I thought, I'm getting too cocky and that may be why I'm getting the wrong values.

So, I reprogrammed the cost function calculation using a **for-loop** over the training examples, where in each iteration I compute the sum over k using a sum() fcn and vector element-wise multiplication. Again, dimensions fit, but the value is wrong. The same value I got on the first try.

Again, I reprogrammed the cost function calculation using **two for-loops** this time, basically transcribing the damn formula into MATLAB, again to no avail! The value I'm getting is the same I got before (5.875588).

So maybe my label encoding is wrong. The way I have it implemented now, is such that I set 1st dimension of the 10-dim vector to 1 if the variable y(i) = 10. Basically, the code is the following:

```
Y = zeros(m, num_labels);
for i = 1:m, Y(i, mod(y(i),num_labels)+1) = 1; end
```

Am I doing this wrong? Or should it be:

```
for i = 1:m, Y(i, y(i)) = 1; end
```

Thanks for any advice

0 · flag

Rob van Putten · a day ago

Hi Jakub, I can imagine the frustration but first of all you really `need to use' vectorization or else it gets very complicated. I made a lot of errors in my code (like using sigmoid instead of sigmoidGradient) but if you read the pdf carefully and use the hints in this topic + regular checks of the matrix sizes it is doable.. but I agree that this is heavy work.. it took me at least 6-8 hours to solve this part..

Good luck!

0 · flag

Rob van Putten · a day ago

perhaps a useable hint as well, do not forget to `translate' all y to vectors (page 5, implementation note). If you are able to do this you can `easily' calculate h;theta(X) using

```
for i = 1:m
   make a vector out of y
   calculate error between y and h;theta(x) for all training sets according to f
ormula in 1.4
end

calculate regularization term using sum(sum(..))
cost = J = 1/m * sum of all errors + regularization term
```

0 · flag

Tom Mosher COMMUNITY TA · a day ago

Hi Jakub,
Your second method seems to work, given that you've created Y of the correct size first. Your first method seems needlessly complicated, so I didn't evaluate it.

This method is even easier, it uses vectorization, so no loop is needed.
https://class.coursera.org/ml-005/forum/thread?thread_id=2091#post-9007

The line of code is automatically repeated for every element in y, by assigning one of the rows of an eye matrix to the output matrix for every row in y.

0 · flag

Jakub Prüher · a day ago

OK! So, I finally figured out where the error was!

I was adding the column of 1's like so:

```
X = [X ones(m,1)];
a2 = [a2 ones(m,1)];
```

When in fact the correct way to do it is this:

```
X = [ones(m,1) X];
a2 = [ones(m,1) a2];
```

After rewriting the respective lines of code my nnCostFunction returns the correct value! My

1st-attempt vectorized implementation was entirely correct, however this seemingly insignificant error has undesirable consequences. Not to mention it is hard to debug, since all the dimensions are correct!

1 · flag

Tom Mosher COMMUNITY TA · 21 hours ago

Excellent point, I'll remember that for future questions from other students.

1 · flag

+ Comment

priyo mustafi · a day ago

Tom, thanks for your tips! Without it I would still be working on the exercise. Very detailed.

1 · flag

Tom Mosher COMMUNITY TA · a day ago

Good work! I'm glad it was useful.

0 · flag

+ Comment

Miguel Almeida · a day ago

I had a different problem than all the ones stated so far. The cause was 50% being overconfident and 50% doing the exercise at 1 AM.

The red flag was a difference of around 0.001 in the gradient checking. If you've ever used gradient checking with complicated formulae (I have), you will know that you do not ALWAYS get values under 1e-9, especially if you don't fine-tune the value of epsilon. So I thought that this could be correct and submitted, only to find that it wasn't correct.

The root cause was that I was computing g'(A2) in step 3, instead of g'(Z2). Which, now that I look at it, is a silly mistake: sigmoids don't look at A's, they look always at Z's. :)

So if you have a difference of 0.001 or in that ballpark when checking your gradient, look here.

1 · flag

Tom Mosher COMMUNITY TA · a day ago

Hi Miguel,
Thanks for your observation. I made a similar error, in computing the regularized cost. I mangled the subscripts of the Thetas, in a way that didn't cause an error, but caused a "small" error. I thought it was close enough, but forgot to submit Part 2, so the error persisted

all the way until I got incorrect results when submitting Part 5. That led me to believe (incorrectly) that my problem was in gradient regularization, overlooking the cost regularization problem. Yes, it was 1am at the time.

0 · flag

+ Comment

Komal Desai · a day ago

I am replying based on the first post.

this is what is written for back-propogation

δ3 equals the difference between a3 and the y_matrix.
δ2 equals the product of δ3 and Θ2 (ignoring the Θ2 bias units), then multiplied element-wise by the g ′() of z2 (computed back in Step 2).

Note that at this point, the instructions in ex4.pdf are specific to looping implementations, so the notation there is different.
Δ2 equals the product of d3 and a2. This step calculates the product and sum of the errors.
Δ1 equals the product of d2 and a1. This step calculates the product and sum of the errors.

how did we jump from $theta3$ to d3?

0 · flag

Tom Mosher COMMUNITY TA · a day ago

Hi Komal,
I am confused. There is no theta3.

0 · flag

+ Comment

Tom Mosher COMMUNITY TA · a day ago

For Darren:
Unfortunately, your post disappeared when Jaideep deleted his thread. That's not very helpful. I'll repeat your post here, hopefully you're monitoring the thread:

I'm getting the same result as bhanu. My a1 is a 16x3 like yours above. Slipping ahead, my a3 is a 16x4 matrix where all the values are between 0 and 1 (as expected). I figure the only thing left that could be wrong is the vectorized implementation of the cost function.

My a2 is 16x5. Did you remember to add the column of bias units?
debug> a2 > 0.5
ans =
1 1 0 1 0
1 1 0 0 1

1 1 0 1 0
1 1 0 1 0
1 0 0 0 1
1 1 0 0 0
1 1 0 1 0
1 0 0 0 1
1 1 0 0 0
1 1 0 1 0
1 0 0 0 1
1 1 0 0 0
1 1 0 1 0
1 1 1 0 0
1 1 0 0 0
1 1 0 1 0

0 · flag

Darren Byrne · a day ago

Hey Tom,

I did. My a2 is the same as yours and my a3 is

0 1 0 1
0 1 1 1
0 1 0 1
0 1 0 1
0 1 1 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1

0 · flag

Tom Mosher COMMUNITY TA · a day ago

That agrees with my a3. Now, it's on to the cost calculation itself. Stand by one while I cook something up...

0 · flag

Tom Mosher COMMUNITY TA · a day ago

Please verify whether your log functions are being used with element-wise multiplication.

There's a big difference between ".*" and "*", but after you take the sum, all of the evidence is hidden.

↑ 1 ↓ · flag

Darren Byrne · a day ago

Yup, that's working, thanks! Feeling pretty thick now... :/

How should we be able to tell when to do an element-wise multiplication as opposed to a matrix multiplication?

↑ 2 ↓ · flag

Tom Mosher COMMUNITY TA · 21 hours ago

If you're using an operator to perform math (like for regular scalar values), and the size of the output should not change, that's a signal to use element-wise operations.

Vector math is used to combine data and transform it using matrix algebra with theta. Those operations are NOT element-wise.

↑ 2 ↓ · flag

+ Comment

□ scroll down for more □